Operációs rendszerek

2016/2017 tavaszi félév

I. ZH segédlet

Cirok Dávid

Nézzük milyen feladattípusok lesznek ZH-n, és hogyan tudunk hozzájuk fogni.

Csővezeték

Ebben a feladattípusban egy egysoros (nyilván parancsértelmező fejlécet leszámítva) scriptet várunk megoldásként. Persze a lényeg az, hogy a script úgy működjön, ahogy kell, szóval ha máshogy oldja meg a hallgató, az is rendben van, de általában a csővezeték a legegyszerűbb hozzáállás. Amit tudunk, csináljunk csővezetékkel.
Hogyan kezdjünk neki? Az ilyen típusú feladatokban valamilyen formában *szűrnünk* kell, ehhez pedig kell egy forrás adathalmaz. Tehát első lépésként valahogy állítsuk elő ezt az adathalmazt. Feladatfüggő, hogy ezt miként tesszük: ha valamilyen input fájlt kell szűrni, akkor a `cat` parancsot használjuk, pl:

```
cat data.csv
```

Ha egy árva stringen szeretnénk valamit szűrni, akkor a `cat` nem fog működni, `echo` kell. Ha környezeti változókat szeretnénk vizsgálni, a `printenv` vagy az `env` parancs fogja őket kiírni. Ha az állományrendszerből kell valami, egyszerűen listázzunk valahogy `ls` -sel (ha nagyon igényesen akarjuk csinálni akkor használuk a `-1` kapcsolót, így szépen látjuk majd a sorokat). Ellenőrizzük, hogy megfelelő-e az adatforrás. A csővezetéket lépésenként építjük, mindig megnézzük a köztes eredményeket és így folytatjuk egyenként.
Most jön az a rész, ahol a szűrőket kezdjük el alkalmazni. Ez nagyban feladatfüggő, fontos, hogy ismerjük a tanult szűrőket, mit mikor mire használunk.
Példa: Írjunk egy scriptet, ami megszámolja, hány egyedi kiterjesztés van a jelenlegi mappában! Oké, szóval az állományrendszerben kell szűrni. Ilyenkor célszerű létrehozni egy rakás próba fájlt (`touch`), hogy ellenőrizhessük magunkat. Nézzük mit látok, ha simán listázok:

```
> ls -1
dsfgfg.html
egyketto.txt
elsofeladat.txt
harmadikfeladat.txt
kiscica.pdf
```

```
kiscica.ppt
kiscica.sh
masodikfeladat.txt
masvalami.txt
negyedikfeladat.txt
```

Ez teljesen megfelel adatforrásnak. Első lépés pipa. A feladat azt kérte, hogy a kiterjesztéseket vizsgáljuk. Mit csinálunk amikor minden sorban van egy rakás információ, de minket csak egy része érdekel? Vágjuk a sorokat `cut` -tal. Ezt csináltuk csv fájloknál is, amikor a sok oszlop közül csak egy-kettő érdekelt minket. Oké, látom a forrást, tudom, hogy a kiterjesztések kellenek, hogy kéne vágni? Pont mentén és a második oszlopot kérem. Ki is próbálom:

```
> ls -1 | cut -d '.' -f 2
html
txt
txt
txt
pdf
ppt
sh
txt
txt
txt
```

Közeledünk. A feladatban egyedi kiterjesztésekről van szó. Az egyediről rögtön ugorjon be a `uniq` parancs. Próbáljuk ki:

```
> ls -1 | cut -d '.' -f 2 | uniq
html
txt
pdf
ppt
sh
txt
```

Ah, majdnem. Kétszer van `txt` . Mi a baj? Eszembe jut, hogy a `uniq` csak az egymást követő ismétlődő sorokat fogja törölni. Hogy tudom orvosolni?

```
> ls -1 | cut -d '.' -f 2 | sort | uniq
html
pdf
ppt
sh
txt
```

Sorba rendeztem őket mielőtt `uniq` -ot futtattam volna rajtuk. Nem felejtjük el, hogy ha számokat rendezünk sorba, használjuk a `-n` kapcsolót!

Már majdnem kész a feladat, ezeknek a számát kell csak megtudnom. Itt jön a képbe a `wc` .

```
> ls -1 | cut -d '.' -f 2 | sort | uniq | wc -l
5
```

Ezzel kész a feladat.

Scriptek írása

Példa 1: Írjunk egy scriptet, ami a paraméterül adott könyvtárban a közönséges fájlokat a Files, a szimbolikus láncokat pedig a Links mappába rendezi!
Nulladik lépés: megijedek a feladattól. Első lépés: elképzelem, milyen vezérléssel tudom megoldani a feladatot. Mivel minden fájlt meg kell vizsgálnom, ezért valószínűleg be kell járnom őket egy `for` ciklussal. Meg is csinálom a keretet a scriptnek:

```
#! /bin/bash

for ... in ...; do
  ...
done
```

Töltsük ki szépen a hiányzó részeket. Az első `...` helyére a ciklusváltozót kell megadnom, ami tetszőleges, legyen mondjuk `fajl` . A második `...` helyére a listát kellene írni, amit bejár a ciklus. Mi kéne, hogy legyen a lista? A paraméterül kapott könyvtárban lévő állományok. Valahogy listázni kellene őket a scripten belül:

```
#! /bin/bash

for fajl in $(ls $1); do
  ...
done
```

Annyit csináltam, hogy parancs behelyettesítéssel listáztam a paraméterül kapott könyvtárat. Nem kérte a feladat, hogy ellenőrizzük a paramétert (van-e egyáltalán paraméter, létezik-e, könyvtár-e), szóval elhisszük, hogy ezzel nem lesz gond, de ezek az ellenőrzések egyszerű `if` ek lennének a script elején. Nézzük, mi kerül a ciklus magjába. Az állományokat, ha közönséges fájlok vagy softlinkek, át kéne helyezni a megfelelő mappába. Ezeket a mappákat létre is kéne hozni először, hogy legyen hova tenni őket. Írjuk is a script elejére ezeket:

```
#! /bin/bash

mkdir $1/Files
mkdir $1/Links
```

```
for fajl in $(ls $1); do
  ...
done
```

Végül írjuk meg a ciklus magját. Minden iterációkor ellenőrzöm az aktuális állomány típusát, és ennek megfelelően áthelyezem valamelyik mappába, vagy nem. Lássuk:

```
#! /bin/bash

mkdir $1/Files
mkdir $1/Links

for fajl in $(ls $1); do
  if [ -f $fajl ]; then
    mv $fajl $1/Files
  elif [ -h $fajl ]; then
    mv $fajl $1/Links
  fi
done
```

Egy megjegyzés a hasonló típusú feladatokhoz: ha könyvtárakat (is) rendezni kellett volna, akkor az általunk létrehozott könyvtárakat át kellett volna ugrani, nehogy azokat is rendezni akarja a script valahova. Ezt egy egyszerű fájlnév ellenőrzéssel tudtuk volna megcsinálni a ciklusmag elején, ha a `$fajl ==` egy általunk éppen létrehozott mappa, akkor `continue`, ugorjunk a következő iterációra. Ennyi volt a feladat.

Példa 2: Írjunk egy scriptet, ami kiírja a jelenlegi mappában lévő .txt fájlok közül azok nevét, amelyek első sora a paraméterül kapott string!
Hasonló lesz az előző feladathoz, be kell járnunk egy mappa tartalmát. Annyi a különbség, hogy most nem minden állomány érdekel minket, hanem csak a .txt fájlok. Nézzük gyorsan át, hogyan tudunk specifikus bejárásokat csinálni:

- “konyvtar” fájljait: `for fajl in $(ls konyvtar);`
- csak “.kit” kiterjesztésű fájlokat: `for fajl in $(ls *.kit);` (ilyen formában a jelenlegi mappában fog keresni)
- a script paramétereit: `for param in $*;`
- szám tartományt: `for i in {1..10};`
- konkrét listát: `for nap in hetfo kedd szerda;`
- …és még sok más lehetőség

Adott a bejárási mód, kezdjük el a scriptet:

```
#! /bin/bash

for fajl in $(ls *.txt); do
```

```
  ...
done
```

Mit kéne csinálni minden iterációban? Vizsgáljuk meg a fájl tartalmát, hasonlítsuk össze az első sort a paraméterben kapott stringgel, és az alapján írjuk ki a fájl nevét, vagy ne. Jó lenne egy változóba elmenteni az aktuális fájl első sorát. Lássuk:

```
#! /bin/bash

for fajl in $(ls *.txt); do
  elsosor=$(cat $fajl | head -1)
  if [ $elsosor == $1 ]; then
    echo "$fajl elso sora $1"
  fi
done
```

Készen is vagyunk a feladattal.

Példa 3: Írjunk egy scriptet, ami kiírja standard kimenetre a szorzótáblát (9*9-ig). Valahogy így:

```
> ./szorzo.sh
  1  2  3  4  5  6  7  8  9
  2  4  6  8 10 12 14 16 18
  3  6  9 12 15 18 21 24 27
  4  8 12 16 20 24 28 32 36
  5 10 15 20 25 30 35 40 45
  6 12 18 24 30 36 42 48 54
  7 14 21 28 35 42 49 56 63
  8 16 24 32 40 48 56 64 72
  9 18 27 36 45 54 63 72 81
```

Mi kellhet ehhez? Próbáljunk meg elgondolkodni rajta mielőtt elolvassuk a megoldást.

Megoldás: két egymásba ágyazott for ciklusban járjuk be 1-től 9-ig a számokat, és szorozzuk őket össze minden körben. Hogyan kezdjünk hozzá? Írjuk le, hogy fog körülbelül kinézni a script:

```
#! /bin/bash

for i in {1..9}; do
  for j in {1..9}; do
    n=$(( $i * $j ))
    echo $n
  done
done
```

Most van 81 szorzatunk, elkezdhetünk gondolkodni a megjelenítésükről. A fenti scriptben csak kiechoztam a számokat, így mindegyik új sorba kerül, nem valami szép. Valahogy rá kéne bírni az `echo` t, hogy ne nyomtasson sortörést minden kiíratáskor. Hm:

```
man echo
```

Nálam rögtön az első paraméter a manuálban az, amit keresek:

> -n do not output the trailing newline

Hurrá. Persze nem akarunk mindent egy sorba, ezért egy üres `echo` kell minden sor után. Ezen kívül, hogy ne direkt egymás mellé tegye a számokat, eléjük teszünk egy üres szóközt is:

```
#! /bin/bash

for i in {1..9}; do
  for j in {1..9}; do
    n=$(( $i * $j ))
    echo -n " $n"
  done
  echo
done
```

Lefuttatom, ezt mondja:

```
 1 2 3 4 5 6 7 8 9
 2 4 6 8 10 12 14 16 18
 3 6 9 12 15 18 21 24 27
 4 8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
```

Majdnem jó. A baj az, hogy vannak egy- és kétjegyű számok is. Hogy kezeljük ezt le? Ellenőrizzük kiíratáskor, hogy kisebb-e a szám tíznél, és ha igen, két szóközt írjunk elé, egyébként egyet.

```
#! /bin/bash

for i in {1..9}; do
  for j in {1..9}; do
    n=$(( $i * $j ))
    if [ $n -lt 10 ]; then
      echo -n "  $n"
    else
      echo -n " $n"
```

```
      fi
    done
    echo
  done
```

Így az output:

```
   1  2  3  4  5  6  7  8  9
   2  4  6  8 10 12 14 16 18
   3  6  9 12 15 18 21 24 27
   4  8 12 16 20 24 28 32 36
   5 10 15 20 25 30 35 40 45
   6 12 18 24 30 36 42 48 54
   7 14 21 28 35 42 49 56 63
   8 16 24 32 40 48 56 64 72
   9 18 27 36 45 54 63 72 81
```